Jurnal Teknologi, 44(D) Jun 2006: 1–14

THE OPTIMIZATION OF STEPPING STONE DETECTION: PACKET CAPTURING STEPS

MOHD NIZAM OMAR[1], MOHD AIZAINI MAAROF[2] & ANAZIDA ZAINAL[3]

Abstract. This paper proposes an optimized packet capturing stone detection algorithm that can shorten the response time of overall response mechanism. The aim of the research is to improve the packet capturing step in stepping stone algorithm, thus, improve the response from overall detection and response system. The proposed method is to use small size of user buffer and kernel buffer. Experiments were conducted with two types of packet stream; i) 10 kbps and ii) 10 000 kbps data generated by Tfgen (packet generator) tools and nine combinations of different buffer sizes for each network packet stream were tested. Results from the experiment were analyzed. From the result, it is proven that the proposed method (by using small size of buffer) gives better result. The research concludes that by using the proposed method, the response time can be improved.

Keyword: IDS, IRS, detecting stepping stones, time gap, optimization

Abstrak. Kertas kerja ini mencadangkan pengoptimuman langkah penawanan paket di dalam algoritma pengesanan batu loncatan yang boleh memendekkan masa tindak balas keseluruhan mekanisma penindakbalasan. Tujuan penyelidikan ini adalah bagi membuktikan langkah penawanan paket di dalam algoritma batu loncatan seterusnya membolehkan percepatan penindakbalasan bahagian penindakbalasan daripada keseluruhan sistem pengesanan dan penindakbalasan. Kaedah yang dicadangkan diperkenalkan dengan menggunakan saiz penimbal pengguna dan penimbal kernel yang kecil. Eksperimen dijalankan dengan menggunakan dua jenis aliran paket rangkaian; i) 10 kbps dan ii) 10 000 kbps yang dijana menggunakan perkakasan Tfgen (penjana paket) dan kombinasi sembilan saiz penimbal yang berbeza untuk setiap aliran paket rangkaian yang diuji. Keputusan daripada eksperimen dianalisa. Daripada keputusan, kaedah yang dicadangkan (dengan menggunakan saiz penimbal yang kecil) memberikan keputusan yang lebih baik. Penyelidikan ini menyimpulkan bahawa dengan menggunakan kaedah yang dicadangkan, masa tindak balas dapat diperbaiki.

Kata kunci: IDS, IRS, pengesanan batu loncatan, jurang masa, pengoptimuman

1.0 INTRODUCTION

IDS can be defined as a system that attempts to identify intrusion, such as unauthorized use, misuses, or abuses of computer systems by either authorized users or external perpetrators [1]. IDS can be divided into two categories, host-based and network-based IDS [2]. From the input perspective, host-based IDS uses logs, system calls, and

[1,2&3] Faculty of Computer Science and Information Systems, Universiti Teknologi Malaysia, 81310 Skudai, Johor. Email: maarofma@fsksm.utm.my, anazida@fsksm.utm.my

so forth while network-based IDS uses network packets as its main input [3]. IRS (Intrusion Response System) can be defined as IDS that detects an attack and immediately responds to remove the intruder from the network [4]. IDS and IRS are related. Both of these systems can use packet capturing program as their main source to detect and respond. IDS detects intrusion while IRS responds after IDS has detected an intrusion. There are many types of responses that IRS can perform such as generating report, locking user account, terminating user session [5], and so on. It is important to note that the success of an attack depends on the time-gap between detection and response [6]. Some efforts to overcome the time-gap problem were accomplished by Ragsdale [7] using adaptive IDS and Foundstone [8] used the preventive approach. In this paper, we focus on the input stage of an IDS. The optimization of both detection and response will give an impact to overall performance; time detection and response processes [9].

Existing NIDS such as Bro [10], Snort and other NIDS use packet capturing software as the tool to obtain information from network. Several packets capturing software or tools such as CMU/Stanford [11], libcap, WinPcap and so on have been used for this purpose. Here, we used WinPcap.

A sample program named CaptureIt was edited from packet capturing program to include a function that has a capability to count the time that has been used for capturing a packet. CaptureIt used WinPcap tools to capture network packet. CaptureIt has been tested with different memory sizes and the time for capturing process was obtained to determine the best buffer size. Besides observing an impact on different sizes of buffer, the CPU and memory usage were also monitored.

1.1 Time Gap

Research by Cohen [12] indicated that the success of an attack depends on the time gap between detection and response. For more comprehensive understanding about this, see the illustration in Figure 1. From the time gap problems, if skilled attackers are given ten hours after they are detected and before a response is made, they will be successful 80% of the time. After thirty hours, the attackers almost never fail [12]. Although Cohen [12] has detected the time gap problem focusing on the time after the intrusion has been detected, we believe that the time detection must also be considered as an important parameter to be optimized. The optimization of both detection and response will give an impact to overall time detection and response processes. Figure 1 shows an effect of optimization processes to overall IDS.

There are many techniques that can be used for an optimization. The technique that will be explained in this paper focuses on the initial step of both detection and response techniques, known as packet capturing technique. The reason for optimizing this packet capturing technique is because it is an initial component used in both detection and response. In addition, it is assumed that optimization of the initial stages in intrusion detection and response is more effective.

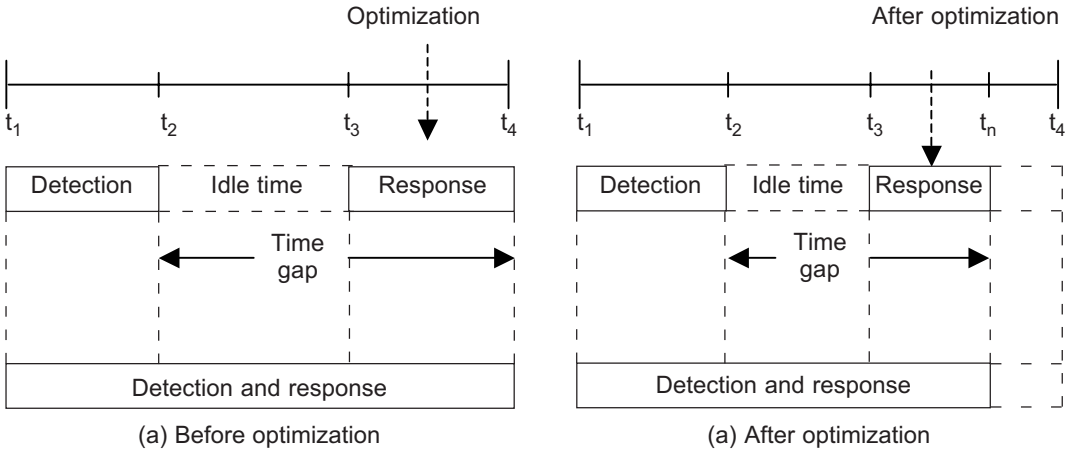

Figure 1 Time gap before and after optimization

1.2 Intrusion Response and Stepping Stone Detection: The Relationship

Tracing intruder was listed by Carver [5] as one of the response techniques. Tracing was chosen because according to Jang [13], tracing technique can identify intruders and prevent them from performing another intrusion. Tracing intruder can be classified [14] into two categories; IP-based and connection-based. Here, we chose to implement connection-based. Unlike IP-based which needs specific hardware in its implementation, connection-based is independent from any hardware and it is also more important than IP-based [14]. Connection-based can be further divided into three areas; host-based, network-based and active network-based [14]. Stepping stone detection falls under network-based category and is currently being researched [15 - 19]. The focus of this research is Stepping Stone Detection. Figure 2 illustrates the relationship between IRS and Stepping Stone Detection. Here, by reducing the time gap (with efficient Stepping Stone Detection algorithm), the response time can be reduced.

1.3 General Steps for Stepping Stones Detection

There are three general steps in Stepping Stone Detection Technique; i) Packet capture, ii) Identification and iii) Comparison. Packet capturing steps are the processes involved where all of the information from raw source (network connection) is obtained and extracted so that the information can be used later. Identification steps are the processes of identifying network connection to obtain its unique identity and finally, comparison steps deal with the comparison of the previously obtained unique identity (identification). Figure 3 shows the general steps of the stepping stone detection.

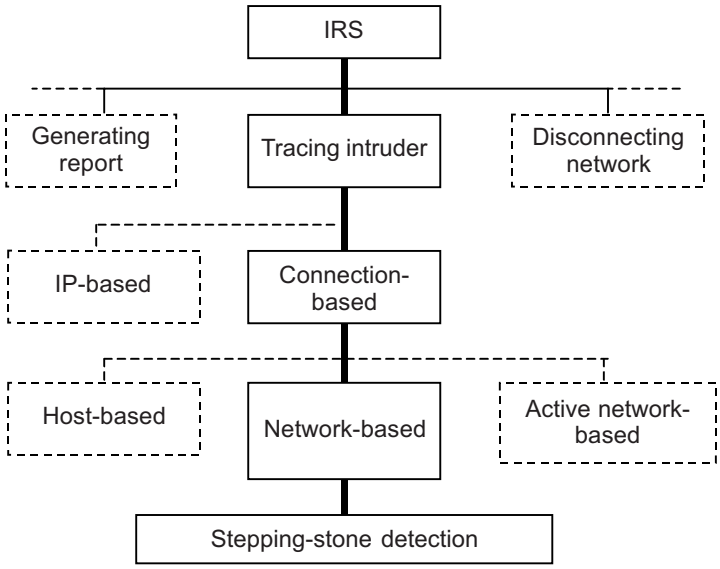

Figure 2 Relationship between intrusion response and stepping stone

Figure 3 General steps of detecting stepping-stones

The research focuses on the first general steps known as packet capturing steps. Detail explanation about packet capturing will be discussed in the next section.

1.4 Packet Capturing

Problem solving requires appropriate diagnostic and analysis tools and ideally, these tools should be available where the problems are located. In Unix environment, to allow such tools to be constructed, a kernel must have some facilities that give a user-level program access to raw, unprocessed network traffic. There are facilities such as NIT [20] in SunOS, the Ultrix Packet Filter [21] in DEC's Ultrix and Snoop in SGI's IRIX, and BSD Packet Filter [22]. All of these facilities only work in Unix-based environment. BSD offers substantial performance improvement over existing packet capture facilities, 10 to 150 times faster than Sun's NIT, and 1.5 times faster than CPSF on the same hardware and traffic mix.

In windows environment, there is a facility called IP filter driver but it is available only for Windows 2000 and it does not support other protocols except IP. It allows controlling and dropping packets but it does not allow monitoring and generating them. PCAUSA offers a commercial product that provides an interface for packet

capture and includes a BPF-compatible filter. However, the user interface is quite low-level and it does not provide abstract function like filter generation.

Most Unix systems provide a set of system calls that allow applications to interact with the network directly. These primitives are useful, for example in packet capture applications, which need to grab the data flowing through the network without any further processing from the kernel. WinPcap is a newly proposed architecture that adds these functionalities to Win32 operating systems. WinPcap includes a set of innovative features that are not available in previous systems [23].

WinPcap is a windows version of BSD Packet Filter. Current IDS such as Bro [10], Snort [24] and others used BSD Packet Filter as their input. Due to the broad usage of BSD Packet Filter in IDS environment, we chose BSD Packet Filter for our experiment. The usage of WinPcap is suitable because WinPcap puts performance as its priority. Experiment done in [23] showed that WinPcap is better than Libpcap (BSD Packet Filter). Furthermore, WinPcap has been proven to be an excellent choice for several applications that are based on high performance packet filtering on Win32 platforms.

1.5 WinPcap

WinPcap is an architecture that is added to the Win32 family of operating system, which enables the system to capture the data of a network using the network adapter of the machine [25]. It provides a high level API to the applications that makes low-level capabilities simpler to use.

1.5.1 Architecture of WinPcap

The basic structure of the WinPcap is given in [25]. WinPcap architecture is made of the following components. NIDS is responsible for the management of the various network adapters. Network Tap [21] is a function to interfere all packets flowing through the network. Filter is used to analyse incoming packet to detect whether it is needed by the user. While kernel buffer is used to transit packets from NIC driver, user buffer is used to transit packets from kernel level. Finally, Packet.dll is used for implementing a set of functions that make communication with the driver simpler [20].

1.5.2 Packet Capture Process

In packet capturing process, the packets will be copied from the NIC driver buffer to the kernel driver buffer and finally, to the user application buffer. In Network Tap, packets that flow through the network are snooped. Later, Filter will analyze incoming packets in order to detect whether a packet is of interest of the user. Consequently, kernel buffer will copy the packet that satisfied the filter. User buffer is then used to store the packets coming from the kernel buffer at user level, with this, it prevents the

application from accessing kernel-managed memory. Application used the packet in user buffer as needed. Our interest is to determine the suitable sizes for WinPcap buffer, kernel buffer, and user buffer. The size of the buffer influences the capturing process, and the maximum amount of data that can be copied from kernel space to user space within a single system call [23].

1.5.3 Kernel Buffer and User Buffer

WinPcap chooses circular buffer as kernel buffer [23]. Circular buffer has been optimized to copy blocks of packets at a time. This buffer allows all the memory to be used to store network bursts. The entire kernel buffer is usually copied by mean of a single read(), thus decreasing the number of system calls and therefore, the number of context switches between user and kernel mode.

Small buffer penalizes the capturing process especially when the application is unable to read as fast as the driver can capture for a limited time interval [23]. The size of the user buffer is very important because it determines the maximum amount of data that can be copied from kernel space to user space within a single system call. Large user buffer causes kernel to wait for the arrival of several packets before copying the data to the user. On the other hand, small value of user buffer means that the kernel will copy the packets as soon as the application is ready to receive them [23]. This paper evaluates the performance of packet capturing process with regards to different buffer sizes.

2.0 TOOLS AND METHODS

This section aims to describe the experiment that has been done using WinPcap. The experiment was intended to investigate packet capturing impact from time perspective by using different buffer sizes. The testbed used in the experiment is shown in Figure 4.

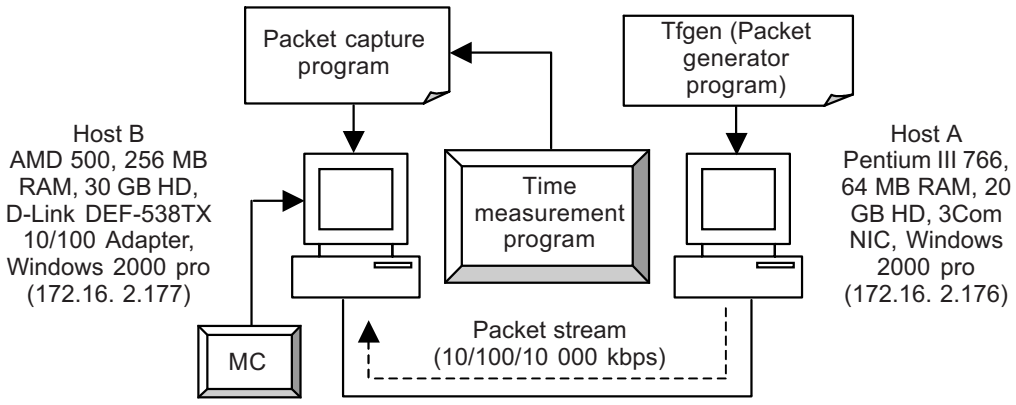

Figure 4 Experiment testbed

Host A is known as Sender and Host B is known as Receiver. This experiment was restricted only to these two hosts as to assure the isolation of the testbed from external sources, therefore, allowing more accurate tests. This same testbed had also been used by [23]. Host A generated traffic using software named Tfgen. The selection of Tfgen as a traffic generator instead of Iperf, Pchar, or Chariot is because Tfgen has many interesting features like being user-friendly, provides GUI, and it can generate multicast traffic.

For each session of experiment, Tfgen was set to the following specifications; utilization rate was 10 kbps and 10 000 kbps, Time to Live was 16, Port was 16, and Traffic Pattern was set to continuous and constant. Host B waited for the packet from Host A. Host B had been equipped with two software; MC (Management Console 1.2) and packet capture program called CaptureIt. CaptureIt contains a function that can be used to capture the time used to capture a packet. In capturing the packet, it uses WinPcap. MC (Management Console 1.2), which is part of Windows 2000 was used to capture the performance during the experiment.

Besides that, we also used a special function to measure the time. It provides six floating point, unlike the clock() function in C language library. Detailed description of the experiment can be found in section 6.1

2.1 Tools

Table 1 lists the equipments used for the experiment.

Table 1 Equipment for the experiment

No	Name	Usage
1	Host A	To generate the packet
2	Host B	To receive the packet
3	Packet capture program	Listen packet
4	Tfgen	Generate packet stream
5	Time measurement program	Capture time that have been used for capturing packet
6	Microsoft Management Console 1.2 (MC)	To administer computer (CPU and memory usage during the experiment)

2.2 Method

The capturing processes were done with different sizes of kernel buffer and user buffer. The size of kernel buffer and user buffer were set to 4, 64, and 1024 MB.

Processes in Figure 5 can be elaborated as follows. As described before, the main purpose of this experiment is to measure the time used for packet capture program to capture a packet. The capturing process was done with different sizes of kernel buffer

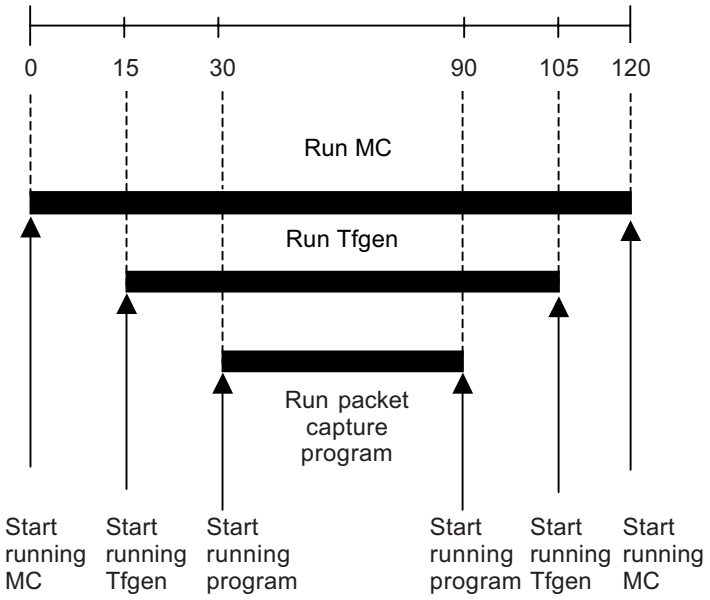

Figure 5 Overall process of an experiment

and user buffer. For each combination sizes (for example 4 MB for kernel buffer and 64 MB for user buffer), experiment was conducted in the order illustrated in Figure 3.

With various combinations of kernel buffer and user buffer sizes, experiment began by running Microsoft Management Console 1.2 (MC). The MC started running with time initialized to zero. 15 seconds after the execution of MC, Tfgen will be executed, then after another 15 seconds, packet capture program was started and it ran for 60 seconds. Time measurement program embedded in packet capture program was used to capture the time used to capture each packet. All these processes continued to run for another 60 seconds. Then packet capture program was stopped. After another 15 seconds, Tfgen was terminated followed by MC in the next 15 seconds. This sequence of running and stopping was changed to another combination of kernel buffer and user buffer sizes.

The important thing in this sequence of testing is time measurement, which was done by packet capture program in 60 seconds interval. From 60 seconds of the running, information on the number of packets successfully being captured, and the time used to capture was recorded (kept in input file) for later usage. Data from the input file was plotted into a graph. The result of this experiment will be discussed in the next section.

3.0 RESULTS AND DISCUSSIONS

Results obtained were in a raw format. Further step was needed to change this raw input. Figure 6 shows the process to obtain the final result.

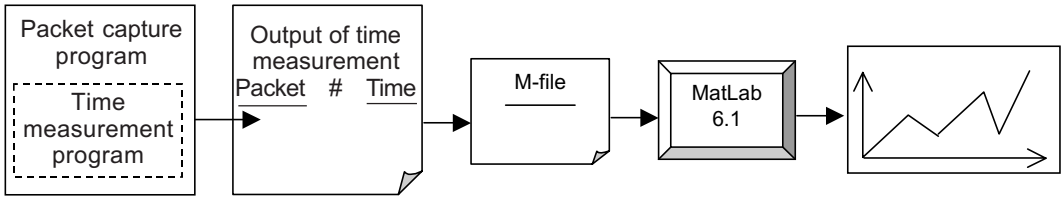

Figure 6 Further step to obtain final experiment output

After an experiment was done for each possible sequence of kernel buffer and user buffer (see Table 2) using steps shown in Figure 5, the next step is to present results in graphs.

In Figure 6, each packet capture program was embedded with time measurement program that will produce output containing number of packets and time spent by packet capture program to execute the capturing process. The output which contained information in list-based style was converted to a graph-based output. We put the raw output into M-file. M-file is a script file which is widely used in MatLab 6.1 software [26]. Then, we plotted graphs as shown in Figures 7, 8, 9, 10, and 11 using Matlab 6.1 software. The final results illustrated in graph-based format are shown in Figures 7, 8, 9, 10, 11 and 12. The outputs are shown from two different sizes of data input, 10 kbps and 10 000 kbps (generated by Tfgen software).

3.1 Experiment Using 10 kbps Data Input

The graph in Figure 7 shows the processing time of the packet capturing using different sizes of kernel and user buffers with 10 Kbps data input of network traffic. The overall

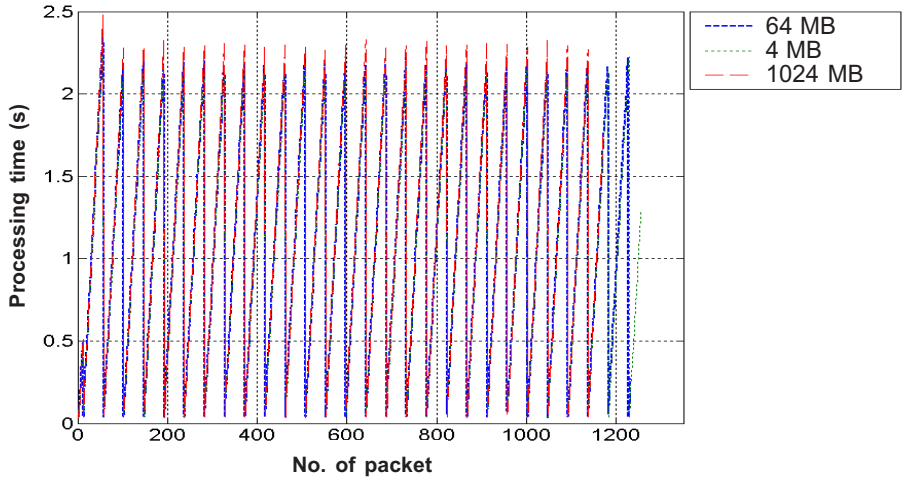

Figure 7 Processing time of packet capturing using 4 MB of kernel buffer and 4, 64 and 1024 MB of user buffer

processing time of packet capturing (regardless of the kernel buffer and user buffer sizes) shows that the processing time is nearly the same for each buffer. The number of packets that are successfully captured by the program is also nearly the same.

Figure 8 shows that the processing time of packet capturing is high when 1024 MB of user buffer is used and low when 4 MB of user buffer is used, while the size of kernel buffer is kept fixed. The highest number of packet that is successfully captured by the program occurred when the small size user buffer is used, and the lowest number of packets is captured when the size of user buffer is 1024 MB.

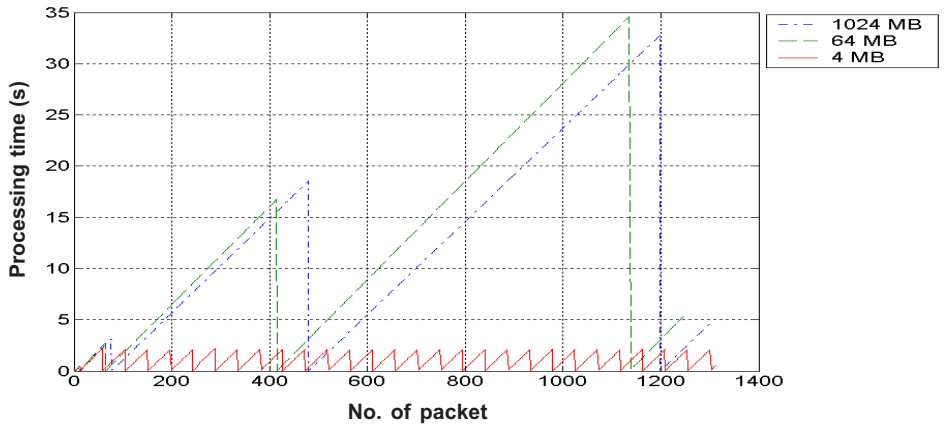

Figure 8 Processing time of packet capturing using 64 MB of kernel buffer and 4, 64 and 1024 MB of user buffer

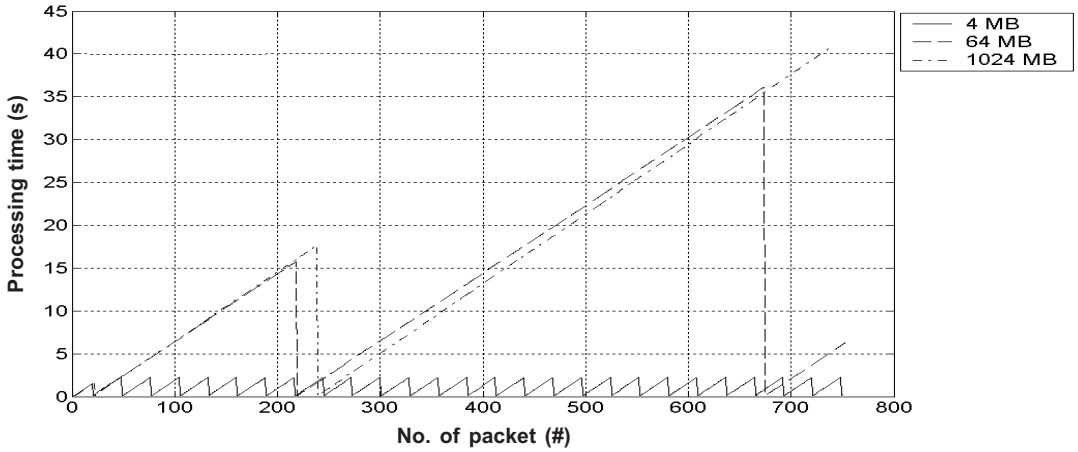

Figure 9 Processing time of packet capturing using 1024 MB of kernel buffer and 4, 64 and 1024 MB of user buffer

Figure 9 reveals that by using 1024 MB of user buffer, the processing time of packet capturing is very high. This is followed by 64 MB of user buffer. When 4 MB of user buffer is used, the processing time is very low. The number of packets that are successfully captured by the program is high when 4 MB of user buffer is used. This is followed by the usage of 64 MB and then, 1024 MB of user buffer size.

From Figures 7, 8 and 9, the usage of small sized buffer, regardless whether it is kernel or user buffer, yields the fastest processing time of packet capturing. It also gives a large amount of packets being captured by the program. The usage of a large buffer size (kernel buffer or user buffer) causes longer capturing time and lower the amount of captured packets.

3.2 Experiment Using 10 000 kbps Data Input

In the next stage, the experiment continued by using 10 000 kbps data input of network traffic. The size of buffer was still the same as the previous experiment (10 kbps data input) except the rate of data input was changed. Figures 10 to 12 depict the results. Figure 10 shows that by using small user buffer size (4 MB), the processing time of packet capturing is faster compared to using large user buffer (1024 MB). Besides, the number of packets successfully captured is higher when small user buffer is used.

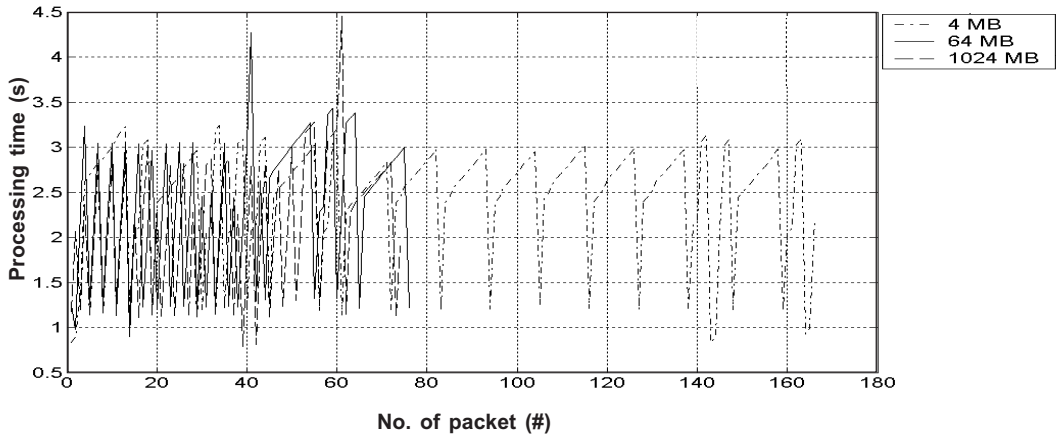

Figure 10 Processing time of packet capturing using 4 MB of kernel buffer and 4, 64 and 1024 MB of user buffer

Figure 11 also shows that when small buffer is used in packet capturing process, the processing time is small. The number of captured packets is also high when small user buffer is used.

Figure 12 shows the same result as the experiment. In here, we used 64 MB kernel buffer. This figure indicates that when small user buffer (4 MB) is used, the packet

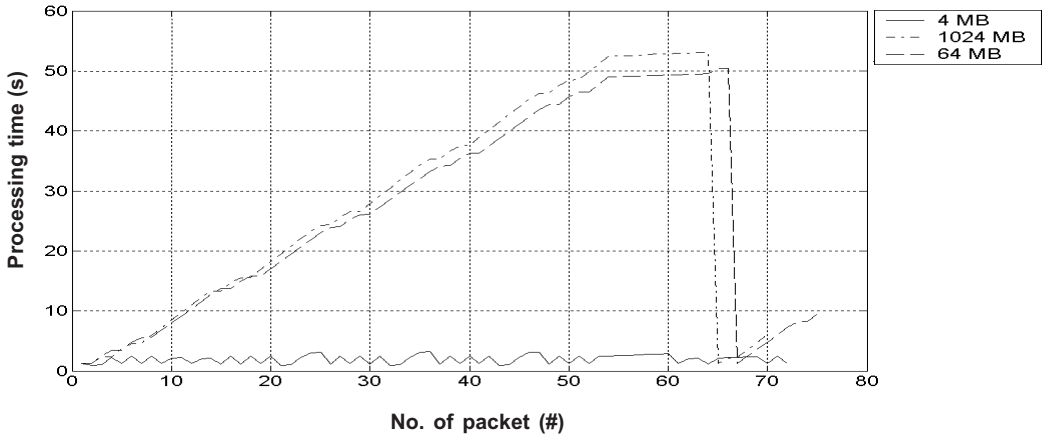

Figure 11 Processing time of packet capturing using 64 MB of kernel buffer and 4, 64 and 1024 MB of user buffer

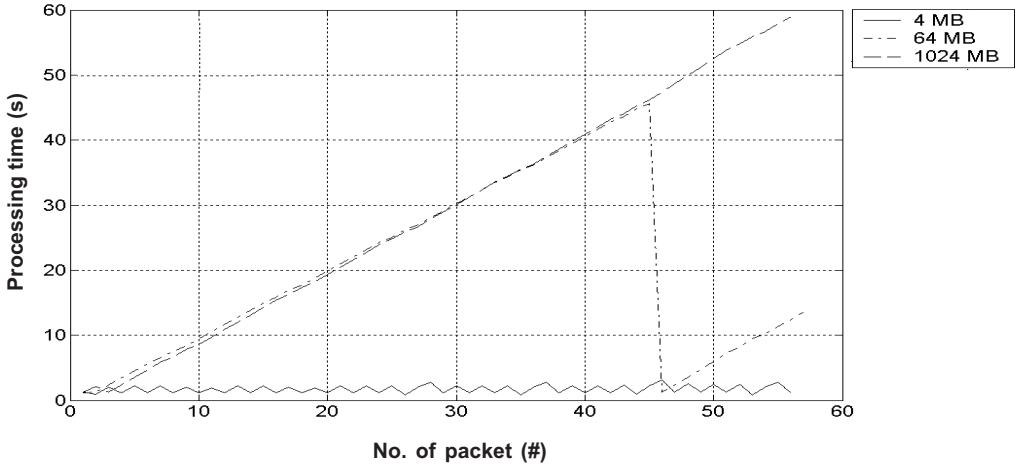

Figure 12 Processing time of packet capturing using 1024 MB of kernel buffer and 4, 64 and 1024 MB of user buffer

capturing time is short. On the other hand, when large user buffer (1024 MB) is used, the result of the processing time is long. The number of the packets successfully captured is nearly the same for different sizes of user buffers.

From Figures 10, 11, and 12, it can be concluded that when a small user buffer is used, the processing time of packet capturing is short. Meanwhile, when a large buffer (1024 MB) is used, a longer packet capturing time is required.

From the different sizes of data input (10 kbps and 10 000 kbps), the conclusion with regards to the buffer size usage and the processing time is the same. Regardless of buffer types, larger size buffer makes the kernel waits for several packets before

copying the data to the user. On the other hand, the small size (4 MB) buffer makes the kernel copies the packet as soon as the application is ready to receive them. In general, we know that packet capturing facilities are widely used in many applications such as network analyzer, packet sniffer and IDS. When these applications used large buffer size (1024 MB), the packet capturing process will be time consuming. In IDS and IRS environment, the delay occurs here can cause the situation known as time gap. To overcome this problem, one of the solutions is to use small size buffer. This is proven through the experiment explained above, a small buffer will produce a faster processing time for packet capturing.

4.0 CONCLUSION

This paper has investigated the relationship between the usage of kernel and user buffers. The experiment has shown that the buffer size does influence the capturing speed and the number of packets successfully captured by packet capturing program. It is found that when small buffer size is used, the number of packets successfully captured in a unit time is larger. By reducing the packet capturing time, the time gap between detection and response time can be reduced [9]. Further works can focus on investigating the side effects of using small buffer size and how to overcome the problem.

REFERENCES

[1] Puketza, N. J., K. Zhang, M. Chung, B. Mukhejee, and R. A. Olsson. 1996. A Methodology for Testing Intrusion Detection System. *IEEE Transactions On Software Engineering*. 22(10): 719-729.

[2] Mukherjee, B., T. L. Heberlein, and K. N. Levit. 1994. Network Intrusion Detection. *IEEE Network*. 8(3): 26-41.

[3] Kerschbaum, F., E. H. Spafford, and D. Zamboni. 2000. Using Embedded Sensors for Detecting Network Attack. Proceeding of the First ACM Workshop on Intrusion Detection Systems. Purdue University, West Lafayette, Indiana.

[4] Kulin, H. T., H. L. Kim, Y. M. Seo, G. Cheo, S. L. Min, and C. S. Kim. 1993. Caller Identification System in the Internet Environment. Proceeding of the USENIX Security Symposium IV. Santa Clara, California, USA.

[5] Carver, A. C. 2002. Intrusion Response Systems: A Survey. Department of Computer Science, Texas A&M University. Collage Station, USA.

[6] Huagang, X. 2000. LIDS Hacking HOWTO, Document for LIDS, v1.0. Linux Intrusion Detection System. http://www.lids.org/lids-howto/lids-hacking-howto.html.

[7] Ragsdale, D. J., C. A. Carver, J. W. Humphries, and U. W. Pooch. 2000. Adaptation Techniques for Intrusion Detection and Intrusion Response System. IEEE International Conference on Systems, Man, and Cybernetics. Nashville, Tennessee, USA.

[8] Founstone, Inc. 1998. Managed Security Service. 2 Venture Street, Suite 100, Irvine, CA 92618.

[9] Omar, M. N., M. A. Maarof, and S. Ibrahim. 2003. Towards Solving Time Gap Problems Through the Optimization of Packet Capture Techniques. CITA '03. Universiti Malaysia Sarawak. Kota Samarahan, Sarawak, Malaysia.

[10] Paxson. V. 1999. Bro: A System for Detecting Network Intruders in Real-Time. *Computer Network*. 31(23-24): 2435-2463.

[11] Mogul, J. C., R. F. Rashid, and M. J. Accetta. 1997. The Packet Filter: An Efficient Mechanism for User-level Network Code. Proceeding of 11th Symposium on Operating Systems Principle. Austin, Texas, ACM.
[12] Cohen. F. B. 1999. Simulating Cyber Attacks, Defenses, and Consequences. Fred & Cohen Associates. http://all.net/journal/ntb/simulate/simulate.html. (accessed in May 1999).
[13] Jang, H., and S. Kim. 2000. A Self-extension Monitoring for Security Management. Computer Security Application, ACSAC'00, 16th Annual Conference. 196-203.
[14] Dong-li, S. 2002. Trend & Techniques of Intruder Traceback. ITU-T Workshop on Security. Seoul, Korea.
[15] Staniford-Chen, S., and L. T. Theberlein. 1995. Holding Intruders Accountable on the Internet. Proceeding of IEEE Symposium on Security and Privacy. Oakland, USA.
[16] Yoda, K. H. 2000. Finding a Connection Chain for Tracing Intruders. In F. Guppens, Y. Deswarte, D. Gollman, and M. Waider. 6th European Symposium on Research in Computer Security – ESORICS 2000 LNCS-1985, Toulouse, France.
[17] Zhang, Y., and V. Paxon. 2000. Detecting Stepping Stone. Proceeding of 9th USENIX Security Symposium. Denver, Colorado.
[18] David, D. L., F. A. Georgina, S. Umesh, P. Vern, C. Jason, and S. Stuard. 2002. Multi Scale Stepping Stone Detection: Detecting Pairs of Jittered Interactive Streams by Exploiting Maximum Tolerable Delay. Fifth International Symposium of Recent Advance in Intrusion Detection. Zurich, Switzerland. Lectures Notes in Computer Science. 2516.
[19] Xinyuan, W., R. S. Douglas, and W. S. Flix. 2003. Inter-Packet Delay Based Correlation for Tracing Encrypted Connection Through Stepping Stones. ESORICS 2003 Symposium. Zurich, Switzerland.
[20] SUN MICROSYSTEMS INC. 1990. NIT(4P), SunOS 4.1.1, Reference Manual. Mountain View, CA. 800-5480.
[21] DIGITAL EQUIPMENT CORPORATION. Packetfilter(4). Ultrix V4.1 Manual.
[22] McCanne, S., and V. Jacobson. 1993 . A New Architecture for User-level Packet Capture. Winter USENIX. 259-270.
[23] Risso, F., and L. Degioanni. 2001. An Architecture for High Performance Network Analysis. Proceedings of Sixth IEEE Symposium on Computers and Communication (ISCC 2001). Hammamet, Tunisia. 686-693.
[24] Northcutt, S., J. Novak, and D. McLachlan. 2000. *Network Intrusion Detection An Analyst's Handbook*. 201, Indianapolis, Indiana: New Riders. 189-190.
[25] Degionni. L. 2000. Development of an Architecture for Packet Capture and Network Traffic Analysis. Master Thesis. Politecnico Di Torino.
[26] Hanselman, D., and B. Littlefield. 1996. *Mastering MATLAB A Comprehensive Tutorial and Reference*. Upper Saddle River, New Jersey: Prentice-Hall.15-16.